

# Numerical Steady-State Solutions of Non-Linear DAE's Arising in RF Communication Circuit Design

Danny Dunlavy\*    Sookhyung Joo†    Runchang Lin‡    Roummel Marcia§  
Aurelia Minut ¶    Jianzhong Sun ||

July 28, 2000

Mentor: Dr. Robert Melville, Lucent Technologies

## 1 Introduction

Large systems of coupled non-linear ordinary differential equations (DAE) arise naturally in applications areas like in the design of radio-frequency integrated circuits. The steady-state response of a non-linear system to periodic or quasi-periodic stimulus is of primary interest to a designer because certain aspects of system performance are easier to characterize and verify in steady state. For example: noise, distortion, blocking are best measured when a circuit is in this state. The system of equations generated in circuit design has the following form,

$$f(v(t)) + \frac{d}{dt}q(v(t)) - b(t) = 0, \quad (1)$$

where  $m$  is the number of circuit nodes excluding the reference,  $q(v(t)) \in \mathbb{R}^m$  is the vector of sums of capacitor charges at each node,  $f(v(t)) \in \mathbb{R}^m$  is the vector of sums of resistor currents at each node,  $b(t) \in \mathbb{R}^m$  is the vector of input currents, and  $v(t) \in \mathbb{R}^m$  is the vector of node voltages. “Closed form” solutions to these ODE’s are extremely difficult, if not impossible, to obtain because of the size of the problem and the complexity of non-linear models. Computing the solutions numerically is a highly effective alternative to computing the solutions analytically.

There are a variety of methods that directly compute the steady-state solution [9]. In the time domain, the standard approaches are finite-difference methods and the shooting methods. Shooting methods solve boundary-value problems by computing the solution to a succession of initial value problems, with steadily improved guesses at an initial condition which results in steady-state. Finite-difference methods generate a sequence of algebraic equations by replacing the system of continuous time differential equations with a discrete system. The sequence of algebraic equations are then solved simultaneously to compute the boundary-value problem solution. Alternatively, in the frequency domain, the unknowns are the coefficients of a

---

\*Department of Mathematics and Statistics, Western Michigan University

†Department of Mathematics, Purdue University

‡Department of Mathematics, Wayne State University

§Department of Mathematics, University of California, San Diego

¶Department of Mathematics, Michigan State University

||Department of Mathematics, Purdue University

trigonometric series [10,11,12]. Such methods applied to nonlinear circuits are referred to as harmonic-balance methods.

Finite-difference methods have not been commonly used for circuits problems because of the large size of the system. Typically, the number of waveforms is between 100 and 1000, and the number of discretization points is between 32 and 1024. Thus the dimension of this system can exceed 100,000(!). Numerical solutions to nonlinear problems typically require the solution of a sequence of linear systems as a subproblem, which for large systems is a computational bottleneck. Recent applications of iterative methods [4-8] to solve the system of linear equations in each Newton step have brought renewed interest in the finite-difference methods. Systems that conventional Gaussian elimination would have floundered on are now manageable because of the memory and time savings possible with such iterative linear solves.

In this report, we solve for the steady-state solution of non-linear DAE's arising from circuit design using finite-difference and harmonic-balancing methods. Various numerical differentiation techniques are considered. Small problems are solved using direct methods while systems of larger size are solved using iterative methods. We apply various preconditioners to speed up the convergence and compare their effectiveness by looking at the number of iterations needed for the iterative method to converge.

## 2 Time-Domain Methods

In the time domain, the initial condition of the steady-state solution must match the solution at the end of one period. Finding the solution to (1) is a two-point boundary-value problem if the solution is required to satisfy

$$v(t_0) = v(t_n) = c.$$

One of the standard approaches for solving boundary-value problems is the finite-difference method.

### 2.1 Numerical Differentiation

Numerical methods for solving ODE's do not produce a continuous approximation to the solution. Rather, approximations are found at certain specified, and often equally spaced, points. Thus, we only require (1) to hold at certain points, say  $\{v(t_0), v(t_1), \dots, v(t_n)\}$ . Since  $q(v(t))$  can be evaluated but not  $\frac{d}{dt}q(v(t))$ ,  $\frac{d}{dt}q(v(t))$  must also be approximated.

There are different ways of approximating the first derivative of a function at a given point. Each one is characterized by how many points are needed to approximate the derivative and how well it approximates the derivative. For example, let  $h = t_k - t_{k-1}$ . By Taylor's Theorem,

$$q(t_{k-1}) = q(t_k) - hq'(t_k) + \frac{h^2}{2}q''(\xi(x)),$$

or

$$q'(t_k) = \frac{q(t_k) - q(t_{k-1})}{h} + \frac{h}{2}q''(\xi(x)).$$

For small values of  $h$ , the difference quotient  $[q(t_k) - q(t_{k-1})]/h$  can be used to approximate  $q'(x_k)$ . This formula is known as the **backward Euler formula**. Since two points are required

to evaluate the backward Euler formula and the error term has a factor of  $h$ , we say that it is a 2-point finite-difference formula of order 1. The following are formulas to approximate the first derivative, the number of points needed to evaluate the formula, and their order.

1. Backward Euler Formula:  $n = 2$ ; order = 1.

$$q'_k = \frac{q(t_k) - q(t_{k-1})}{h}.$$

2. Forward Euler Formula:  $n = 2$ ; order = 1.

$$q'_k = \frac{q(t_{k+1}) - q(t_k)}{h}.$$

3. BDF:  $n = 3$ ; order = 2.

$$q'_k = \frac{\frac{3}{2}q(t_k) - 2q(t_{k-1}) + \frac{1}{2}q(t_{k-2})}{2h}$$

4. Centered-Difference Formula:  $n = 2$ ; order = 2.

$$q'_k = \frac{q(t_{k+1}) - q(t_{k-1}))}{2h}.$$

For the backward Euler formula, we can write an approximation to (1) as

$$\begin{bmatrix} f_1(v_0) \\ f_1(v_1) \\ \vdots \\ f_1(v_{n-1}) \\ f_1(v_n) \\ \vdots \\ f_m(v_0) \\ f_m(v_1) \\ \vdots \\ f_m(v_{n-1}) \\ f_m(v_n) \end{bmatrix} + \frac{1}{h} \begin{bmatrix} 1 & 0 & \dots & -1 \\ -1 & 1 & \ddots & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & -1 & 1 \\ & & & \ddots \\ & & & & 1 & 0 & \dots & -1 \\ & & & & -1 & 1 & \ddots & 0 \\ & & & & 0 & \ddots & \ddots & 0 \\ & & & & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} q_1(v_0) \\ q_1(v_1) \\ \vdots \\ q_1(v_{n-1}) \\ q_1(v_n) \\ \vdots \\ q_m(v_0) \\ q_m(v_1) \\ \vdots \\ q_m(v_{n-1}) \\ q_m(v_n) \end{bmatrix} - \begin{bmatrix} b_1(t_0) \\ b_1(t_1) \\ \vdots \\ b_1(t_{n-1}) \\ b_1(t_n) \\ \vdots \\ b_m(t_0) \\ b_m(t_1) \\ \vdots \\ b_m(t_{n-1}) \\ b_m(t_n) \end{bmatrix} = 0, \quad (2)$$

where  $v_i = v(t_i)$ . The zero of this function would correspond to the approximate values of  $v(t)$  at times  $\{t_0, t_1, \dots, t_n\}$  which represent a steady-state solution. We solve this numerically by applying Newton's method.

## 2.2 Newton's Method

Because of its quadratic convergence near a solution, Newton's method is one of the most powerful and well-known numerical methods for solving a zero-finding problem  $H(x) = 0$ , where  $H : \mathbb{R}^p \rightarrow \mathbb{R}^p$ . It generates a sequence of iterates that converge to a solution by solving a sequence of systems of linear equations. A typical implementation of Newton's method with a linesearch is the following:

```

Initialize  $x \leftarrow x_0$ ;
Compute  $H(x)$ ;
do while  $\|H(x)\| > \epsilon$ ;
    Compute  $DH(x)$ ;
    Solve  $DH(x)\Delta x = -H(x)$ ;
    Choose steplength  $\sigma \in (0, 1]$ ;
     $x \leftarrow x + \sigma\Delta x$ ;
end do

```

where  $DH(x)$  is the Jacobian of  $H(x)$ .

### 2.2.1 Linesearch

Newton's method works under the assumption that

$$H(x_{k+1}) \approx H(x_k) + DH(x_k)(x_{k+1} - x_k).$$

It finds the zero of the right-hand side to compute the step. If  $H(x)$  is badly behaved, e.g., changes in the derivatives are big, then this truncated first-order Taylor expansion is a poor approximation of  $H(x)$  near  $x_k$ . Thus by taking a step along  $\Delta x$ , the iterates might actually move away from  $x^*$ , the zero of the function, and Newton's method might not converge. Also, by choosing  $x_{k+1} = x_k + \Delta x$ , there is no guarantee that  $\|H(x_{k+1})\|_2 \leq \|H(x_k)\|_2$ . To safeguard the iterates from diverging and guarantee that the function value decreases at each iterate, the steplength along every  $\Delta x$  is limited by an appropriate criterion.

Linesearch techniques determine a steplength  $\sigma \in (0, 1]$ , along  $\Delta x$  that ensures a decrease in the norm of the function value and helps Newton method converge. In our implementation of the linesearch,  $\sigma$  is chosen such that

$$\|H(x_k + \sigma\Delta x)\|_2 \leq \|H(x_k + \tau\Delta x)\|_2$$

for all  $\tau \in [0, 1]$ . Such a  $\sigma$  always exists since  $\Delta x$  is a descent direction for the function  $\|H(x)\|_2$ . By choosing  $\sigma$  this way, we compute  $x_{k+1}$  such that  $\|H(x_{k+1})\|_2 \leq \|H(x_k)\|_2$  while maximizing the descent along the full Newton step  $\Delta x$  for each iteration. Near the zero of the function, note that linesearch steplength approaches the value 1 since the linear Taylor expansion better approximates the function.

### 2.2.2 Reverse Call

A feature called reverse call is used in our implementation of Newton's method. It requires that all function evaluations, matrix-vector multiplications, and linear solves be done in the main file. In this way, the Newton solver is completely independent of the calling sequence for all the necessary operations for solving

$$DH(x)\Delta x = -H(x).$$

This becomes crucial when applying the Newton solver to problems with different data structures. In Newton solvers without reverse call, the calling sequence for performing the linear solves must be altered for problems with different data structures. Thus the Newton solver must be modified. On the other hand, Newton solvers with reverse call need not be changed for problems with different data structures. The appropriate changes are made in the main file which must be changed anyway for different problems.

### 2.2.3 Tensor Products

Tensor products for matrices are also known as Kroneker products [3]. Denoted by  $\otimes$ , it forms an  $mn \times mn$  matrix  $A \otimes B$  from an  $m \times m$  matrix  $A$  and an  $n \times n$  matrix  $B$  by replacing each component  $a_{ij}$  of  $A$  by the  $n \times n$  matrix  $a_{ij}B$ , where  $1 \leq i \leq m$ , and  $1 \leq j \leq m$ . For example, suppose  $A$  and  $B$  are two matrices, where

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \quad \text{then } A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{bmatrix}.$$

i.e.  $A \otimes B$  will stamp four copies of  $B$  and scale each copy by the corresponding scalar element of  $A$ . The backward Euler difference matrix in (2) can be written as  $(I_m \otimes \Delta_n)$ , where

$$\Delta_n = \begin{bmatrix} 1 & 0 & \dots & 0 & -1 \\ -1 & 1 & 0 & \ddots & 0 \\ 0 & -1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 & 0 \\ 0 & \dots & 0 & -1 & 1 \end{bmatrix}.$$

Define  $H(v) : \mathbb{R}^{mn} \rightarrow \mathbb{R}^{mn}$  as the left hand side of 2. We write  $H(v)$  as <sup>1</sup>.

$$H(v) = \begin{bmatrix} f() & & \\ & \ddots & \\ & & f() \end{bmatrix} v + \frac{1}{h} (\Delta_n \otimes I_m) \begin{bmatrix} q() & & \\ & \ddots & \\ & & q() \end{bmatrix} v - b,$$

or by an abuse of notation,

$$H(v) = (I_n \otimes f())v + \frac{1}{h} (\Delta_n \otimes I_m)(I_n \otimes q())v - b.$$

### 2.2.4 Stride Permutations

While tensor product allows us to represent large matrices of a certain structure in compact form, stride permutations allow us to greatly simplify our implementation of tensor product actions on vectors. Simply put, a stride permutation maps an  $mn$ -vector to an  $nm$ -vector in the following way:

$$\begin{bmatrix} u_{11} \\ \vdots \\ u_{1n} \\ u_{21} \\ \vdots \\ u_{2n} \\ \vdots \\ u_{m1} \\ \vdots \\ u_{mn} \end{bmatrix} \leftrightarrow \begin{bmatrix} u_{11} \\ \vdots \\ u_{n1} \\ u_{12} \\ \vdots \\ u_{n2} \\ \vdots \\ u_{1m} \\ \vdots \\ u_{nm} \end{bmatrix}.$$

---

<sup>1</sup>The pseudo-matrix  $[f()]$  was suggested by Peter Feldman

Note that on the level of 2-dimensional arrays, stride permutation is equivalent to matrix transpose. Tensor products and stride permutations are closely linked in the following way:

**Theorem** If  $A$  and  $B$  are  $mn \times mn$  matrices and  $P$  is a stride permutation, then

$$P(A \otimes B)P^{-1} = (B \otimes A).$$

**Proof:** [3].

Consider the action of the backward Euler difference matrix  $(\Delta_n \otimes I_m)$  on the vector  $(I_n \otimes q())v$ . By applying the stride permutation  $P$ , we obtain

$$\begin{aligned} P^{-1}P(\Delta_n \otimes I_m)P^{-1}P(I_n \otimes q())v &= P^{-1}(I_m \otimes \Delta_n)P(I_n \otimes q())v \\ &= P^{-1} \begin{bmatrix} \Delta_n & & \\ & \ddots & \\ & & \Delta_n \end{bmatrix} P(I_n \otimes q())v. \end{aligned}$$

The system is now decoupled. Hence computing the product is made easy.

### 2.2.5 Constructing the Jacobian

With this notation, we can now construct the Jacobian of the discretized DAE:

$$H(v) = (I_n \otimes f())v + \frac{1}{h}(\Delta_n \otimes I_m)(I_n \otimes q())v - b.$$

Since the differential operator is linear, then

$$DH(v) = \begin{bmatrix} Df() & & \\ & \ddots & \\ & & Df() \end{bmatrix} v + \frac{1}{h}(\Delta_n \otimes I_m) \begin{bmatrix} Dq() & & \\ & \ddots & \\ & & Dq() \end{bmatrix} v,$$

or

$$DH(v) = (I_n \otimes Df())v + \frac{1}{h}(\Delta_n \otimes I_m)(I_n \otimes Dq())v.$$

Note that the Jacobian is sparse and hence does not need too much computational memory. Not only that, performing matrix-vector products with the Jacobian can be made faster using stride permutations. These features will play a prominent role in one of the methods for solving the Newton equations.

The bottleneck in Newton's method is solving the linear system

$$DH(x)\Delta x = -H(x).$$

We now consider the two main methods for solving these equations.

### 2.3 Direct Methods

Direct techniques for solving a system of linear equations  $Ax = b$  are methods that give an answer in a fixed number of steps, subject only to roundoff errors. For arbitrary nonsingular matrices  $A$ , Gaussian elimination is the principal tool in the direct solution of such systems. It computes a lower triangular matrix  $L$  and an upper triangular matrix  $U$  such that

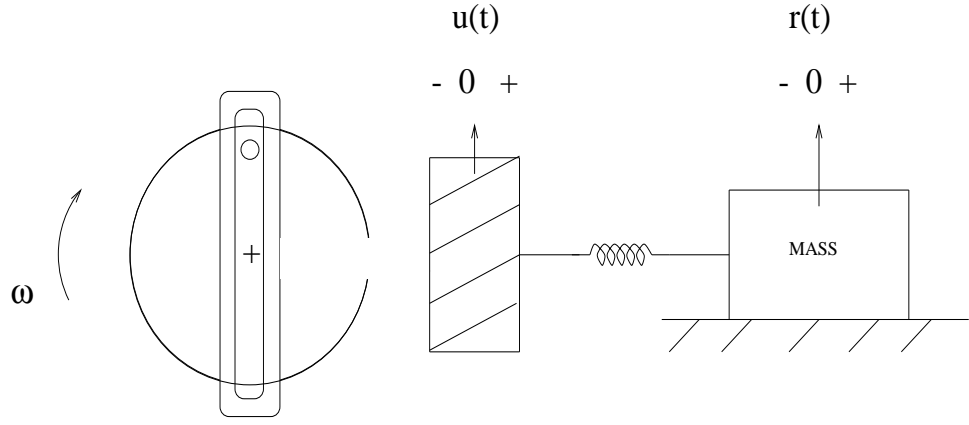
$$A = LU.$$

(Often pivoting is used to stabilize the algorithm. The factorization  $PA = LU$  is what is actually computed.) Solving  $Ax = b$  then reduces to solving two triangular systems

$$Ly = b, \quad Ux = y. \quad (3)$$

Since  $L$  and  $U$  are triangular matrices, (3) is easily solved using back and forward substitution. In our implementation, the subroutine `dgetrf` in LAPACK is used to compute the  $LU$  factorization of  $A$  and `dgetrs` to solve  $Ly = b$  and  $Ux = y$ .

**Example 1.** Consider the following mechanical resonance example:



where  $u(t)$  is the driving term,  $r(t)$  is the displacement term, and  $s(t)$  is the time derivative of  $r(t)$ . This mechanical system is described by the following equations:

$$\begin{aligned} u(t) - \text{Ampl} \cdot \sin(\omega t) &= 0 \\ s(t) - \frac{d}{dt}r(t) &= 0 \\ \text{Mass} \cdot \frac{d}{dt}s(t) + \text{Damp} \cdot s(t) + \text{Spring}(r(t) - u(t)) &= 0, \end{aligned}$$

We can write these in the form (1), where

$$f : \begin{bmatrix} u \\ r \\ s \end{bmatrix} \rightarrow \begin{bmatrix} u \\ s \\ \text{Damp} \cdot s + \text{Spring}(r - s) \end{bmatrix}, \quad q : \begin{bmatrix} u \\ r \\ s \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ -r \\ \text{Mass} \cdot s \end{bmatrix}, \quad b(t) = \begin{bmatrix} \text{Ampl} \cdot \sin(\omega t) \\ 0 \\ 0 \end{bmatrix}.$$

If the spring term in the third equation is linear, then this DAE have the following solution:

$$\begin{aligned}
 u(t) &= A \sin(\omega t) \\
 r(t) &= c_1 e^{-Dt/2M} \cos\left(\frac{\sqrt{D^2 - 4Mk}}{2M} t\right) + c_2 e^{-Dt/2M} \sin\left(\frac{\sqrt{D^2 - 4Mk}}{2M} t\right) \\
 &\quad + \frac{kA(k - M\omega^2)}{(k - M\omega^2)^2 + (D\omega^2)} \sin(\omega t) + \frac{-kAD\omega}{(k - M\omega^2)^2 + (D\omega^2)} \cos(\omega t) \\
 s(t) &= \frac{d}{dt} r(t),
 \end{aligned}$$

where  $A$  = Ampl,  $M$  = Mass,  $D$  = Damp, and  $k$  = Spring.

We now compare the analytic solution with a numerical solution. For this problem  $A = 1$ ,  $M = 1$ ,  $D = 0.2$ , and  $\omega = 1/6$ . The backward Euler was used to approximate the derivative

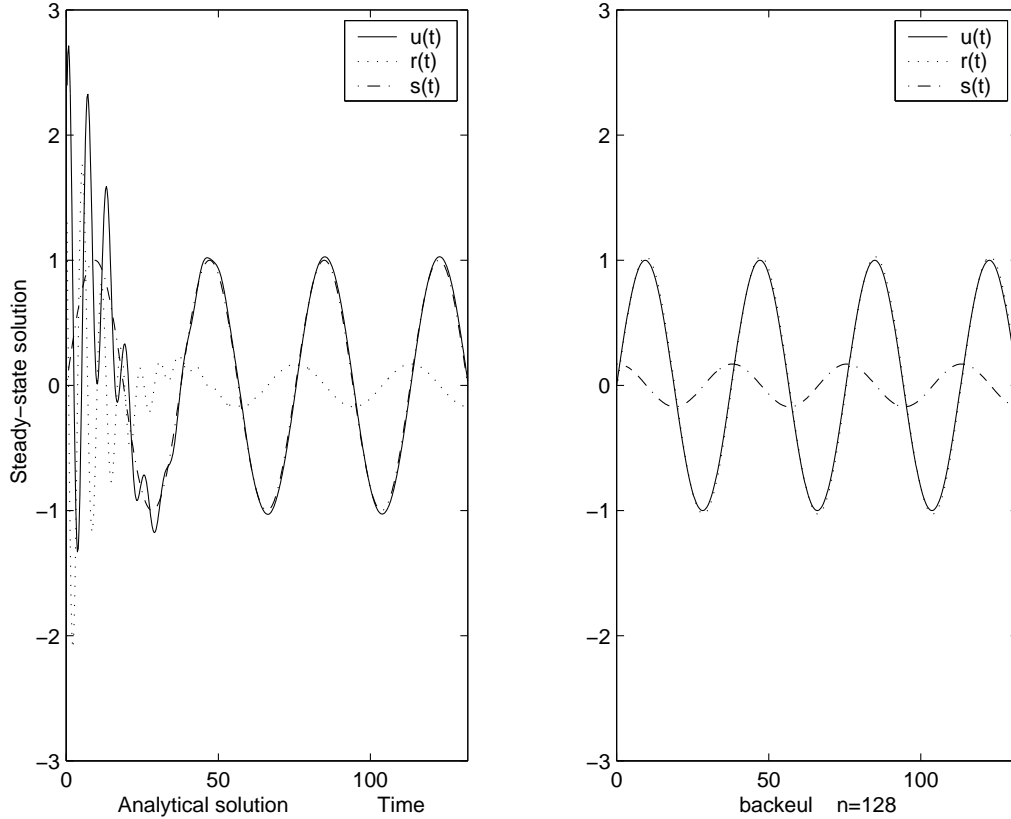


Figure 1: comparison between numerical soln and analytic soln

Note that the analytic solution becomes nearly steady after one period. Other forms of differentiation operators were also tried with equal success.

Figure (2) shows the norm of the amplitude of a response to a fixed damping at various excitation frequencies. The overlap and discontinuity suggests that there exists multiple steady states with one unstable solution which points out a qualitative difference between the linear and non-linear cases.

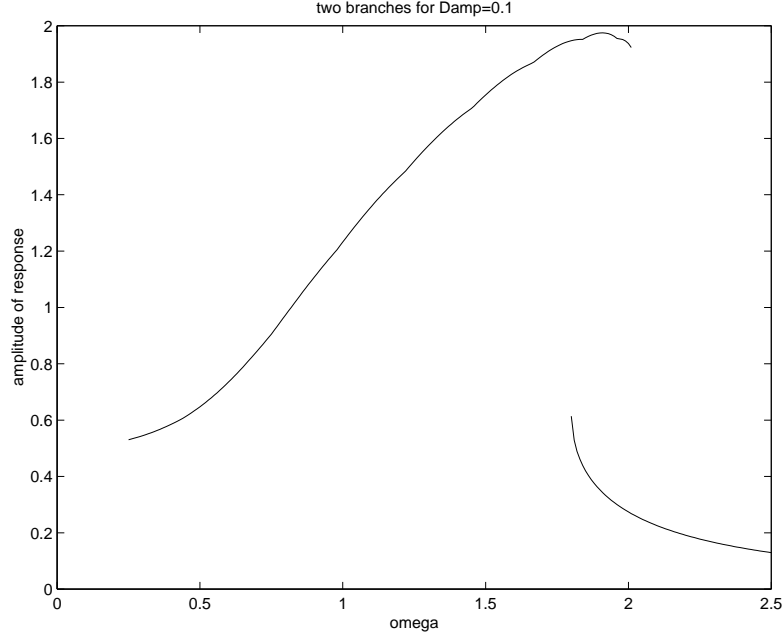


Figure 2: Amplitude of response for fixed damp = 0.1

## 2.4 Iterative Methods

For a large sparse matrix  $A$  of dimension  $p \geq 10,000$ , computing its  $LU$  decomposition might not be suitable for a couple of reasons. First, although  $A$  is sparse,  $L$  and  $U$  might not be. Thus a full  $p \times p$  matrix must be stored. Memory in this case becomes an issue since  $n^2$  entries must be stored. Second the computational time it takes to form such factorizations is  $O(p^3)$ . If solving systems of this size many times is necessary, it is easy to see why forming the  $LU$  factors would be cumbersome. An alternative to solving  $Ax = b$  directly is to use iterative methods.

Iterative methods to solve the linear system  $Ax = b$  start with an initial approximation  $x_0$  to the solution  $x^*$  and generates a sequence of vectors  $\{x_k\}_{k=0}^{\infty}$  that converge to  $x^*$ . For large sparse systems, these techniques are efficient in terms of both computer storage and computational time since the matrices do not have to be formed explicitly and are used only in matrix-vector operations. The matrix fill-ins that occur in direct factorization does not happen in iterative methods.

### 2.4.1 Preconditioning

The number of iterations for an iterative method to converge depends on the number of distinct eigenvalues of  $A$ . In general, the closer the eigenvalues of  $A$  to unity, the fewer iterations it takes to converge. When  $A$  has very scattered eigenvalues, solving the linear system iteratively might just be as slow as solving it directly. A technique known as preconditioning can be applied to alleviate this problem.

Since iterative methods converge faster whenever the eigenvalues of  $A$  are close to one, instead of solving the system  $Ax = b$ , the system

$$\tilde{A}^{-1}Ax = \tilde{A}^{-1}b$$

where  $\tilde{A} \approx A$ , is solved.  $\tilde{A}$  is called a (left-) *preconditioner* of the system  $Ax = b$ . The effectiveness of  $\tilde{A}$  depends on how well the eigenvalues of  $\tilde{A}^{-1}A$  are clustered and how easy it is to solve the linear system  $\tilde{A}y = c$ . Typical preconditioners are diagonal and triangular matrices. There is not one preconditioner that can be used for all linear systems. The best preconditioners are typically derived from the applications where the linear system arises.

### 2.4.2 Circulant Matrices

Circulant matrices are matrices of the form

$$C = \text{circ}\left(\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix}\right) = \begin{bmatrix} c_0 & c_n & c_{n-1} & \dots & c_1 \\ c_1 & c_0 & c_n & \dots & c_2 \\ c_2 & c_1 & \ddots & \ddots & c_3 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ c_n & c_{n-1} & \dots & c_1 & c_0 \end{bmatrix}.$$

For example, the matrices for all the numerical differentiation are circulant matrices. Circulant matrices have the property that as a class of matrices, they are all simultaneously diagonalized by the discrete Fourier transform (DFT)

$$F_n = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{(n-1)2} & \dots & \omega^{(n-1)^2} \end{bmatrix},$$

where  $\omega$  is the  $n$ th root of unity. For example, for  $n = 4$ , if  $j$  is the fourth root of unity, then

$$\begin{bmatrix} a & d & c & b \\ b & a & d & c \\ c & b & a & d \\ d & c & b & a \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & j^2 & j^3 \\ 1 & j^2 & j^4 & j^6 \\ 1 & j^3 & j^6 & j^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & j^2 & j^3 \\ 1 & j^2 & j^4 & j^6 \\ 1 & j^3 & j^6 & j^9 \end{bmatrix} \begin{bmatrix} \lambda_0 & & & \\ & \lambda_1 & & \\ & & \lambda_2 & \\ & & & \lambda_3 \end{bmatrix}.$$

Note that this also implies that diagonal matrices under DFT similarity transformations are circulant. Note also that linear circulant systems are easy to solve and can be done in  $O(p \log p)$  steps. The class of circulant matrices has many other properties such as closure under addition, multiplication, and inverses, but these properties are not used in our context.

It is not hard to see that for the block diagonal matrices

$$\begin{bmatrix} Df() & & & \\ & Df() & & \\ & & \ddots & \\ & & & Df() \end{bmatrix} \text{ and } \begin{bmatrix} Dq() & & & \\ & Dq() & & \\ & & \ddots & \\ & & & Dq() \end{bmatrix},$$

their image under the similarity transformation  $(F_n \otimes I_m)$  are circulant. Similarly, the image of  $P(\Delta_n \otimes I_m)P^{-1}$  where  $P$  is the stride permutation, under  $(F_n \otimes I_m)$  is block diagonal. Thus

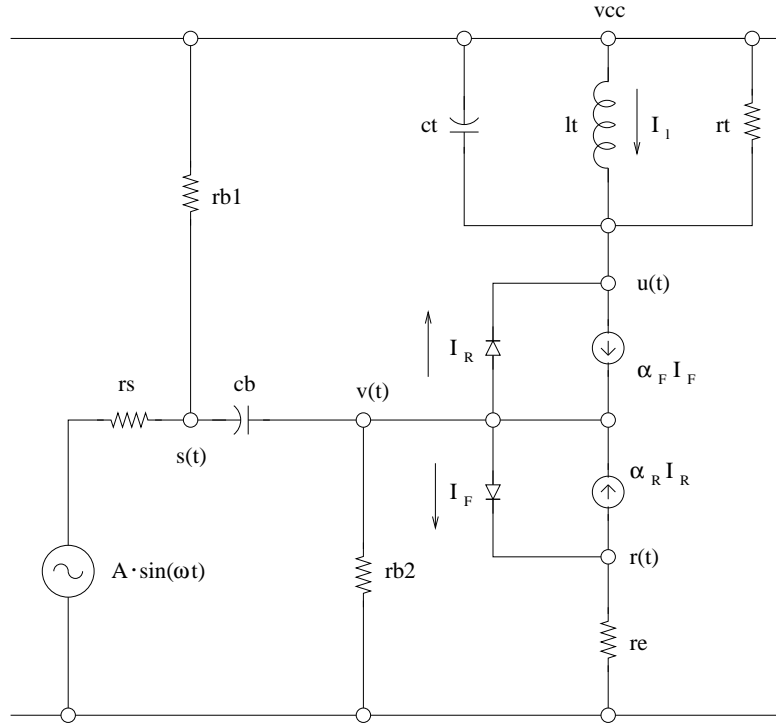
the matrix

$$H(x) = (F_n \otimes I_m)(I_n \otimes Df())(F_n^* \otimes I_m) + \frac{1}{h}(F_n \otimes I_m)(\Delta_n \otimes I_m)(F_n^* \otimes I_m)(F_n \otimes I_m)(I_n \otimes Dq())(F_n^* \otimes I_m).$$

is the sum of a block circulant matrix and a block-scaled circulant matrix. It has been observed that this matrix sum is often block-diagonally heavy. We attempt to use this as a preconditioner in the time domain by applying inverse DFT similarity transformations and compare its effectiveness with the other preconditioners.

### 2.4.3 Circuits Problem

Consider the following circuit example. The system equations for the above amplifier are:



$$\begin{aligned}
ct \frac{d}{dt}(vcc - u(t)) + I_R + I + \frac{vcc - u}{rt} - \alpha_F I_F &= 0 \\
cb \frac{d}{dt}(s - v) + \frac{vcc - v}{rb_1} - \frac{v}{rb_2} - I_R - I_F + \alpha_F I_F + \alpha_R I_R &= 0 \\
I_F - \frac{r}{re} - \alpha_R I_R &= 0 \\
\frac{A \cdot \sin(\omega t) - s}{rs} - cb \frac{d}{dt}(s - v) &= 0 \\
lt \frac{d}{dt}I - (vcc - u) &= 0 \\
I_F - \text{dio}(v - r) &= 0 \\
I_R - \text{dio}(v - u) &= 0,
\end{aligned}$$

where

$$\begin{aligned}
\omega &= 84 \times 10^6 \times 2\pi & A &= 0.5 \\
rb_1 &= 4.7 \times 10^3 & rb_2 &= 700 \\
re &= 5 & rt &= 200 \\
rs &= 50 & lt &= 92 \times 10^{-9} \\
cb &= 1 \times 10^{-6} & ct &= 39 \times 10^{-12} \\
vcc &= 8 \\
\alpha_F &= 0.98 & \alpha_R &= 0.25
\end{aligned}$$

and  $\text{dio}(y) = Is e^{rVt} y - 1$  where  $Is = 1 \times 10^{-16}$  and  $rVt = \frac{1}{0.0257028}$ . We first formulate the above circuit to the standard form as in (1), i.e.,

$$\mathbf{f}(\mathbf{x}(t)) = \begin{bmatrix} I_R + I + \frac{vcc - u}{rt} - \alpha_F I_F \\ \frac{vcc - v}{rb_1} - \frac{v}{rb_2} - I_R - I_F + \alpha_F I_F + \alpha_R I_R \\ I_F - \frac{r}{re} - \alpha_R I_R \\ -\frac{s}{rs} \\ -(vcc - u) \\ I_F - \text{dio}(v - r) \\ I_R - \text{dio}(v - u) \end{bmatrix}, \quad \mathbf{q}(\mathbf{x}(t)) = \begin{bmatrix} ct \frac{d}{dt}(vcc - u(t)) \\ cb \frac{d}{dt}(s - v) \\ 0 \\ -cb \frac{d}{dt}(s - v) \\ lt \frac{d}{dt}I \\ 0 \\ 0 \end{bmatrix},$$

$$\mathbf{b}(t) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{A \cdot \sin(\omega t)}{rs} \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{x}(t) = \begin{bmatrix} u(t) \\ v(t) \\ r(t) \\ s(t) \\ I(t) \\ I_F(t) \\ I_R(t) \end{bmatrix}.$$

In this example, the number of equations  $m$  is 7. Discretizing one period into  $n = 100$  nodes, we obtain the following results using the backward Euler formula and Gaussian elimination to solve the linear subproblem:

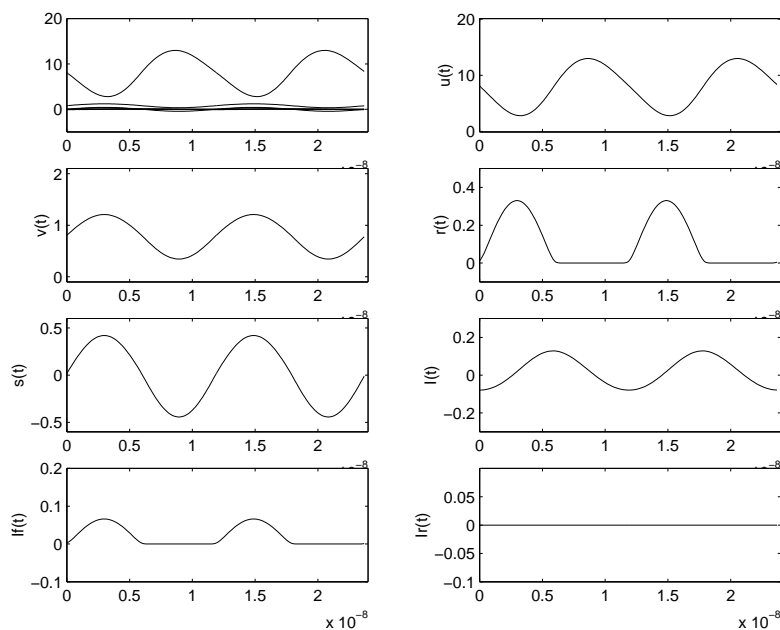


Figure 3: Solutions of Amplifier

The graphs are taken over two periods. The initial value is chosen to be the vector of all ones. The error tolerance  $10^{-10}$  is reached after 10 iterations. Note that the waveform of the response of  $r(t)$  and  $I_F(t)$  are non-sinusoidal.

#### 2.4.4 Preconditioner Comparisons

The rate of convergence can be dramatically affected by the choice of a preconditioner. Some of the possibilities are discussed here.

Preconditioner	Differentiators			
	BDF2	BE	CD	FE
Band-diag( $\frac{\partial H}{\partial x}$ )	6	4	na	5
Block-diag( $F \frac{\partial H}{\partial x} F^*$ )	20	21	22	49
Diag( $g_0, g_1, \dots, g_{n-1}$ )	94	94	63	94
Block-diag( $\frac{\partial H}{\partial x}$ )	99	97	na	97
Identity	149	146	143	150

Table 1: Number of time-domain iterations using preconditioners and 4 differential operators

The choice of a good preconditioner depends on the structure of the Jacobian matrix, which in turn, depends on the choice of the differential operator. For the BDF2, backward Euler (BE) and forward Euler (FE) differentiation, the Jacobian matrix is an upper or lower block bidiagonal matrix with a coupling block on one corner. The above Jacobian matrices are given in

The first preconditioner is then taken to be the band-diagonal Jacobian matrix with the coupling block removed. However, this choice is not appropriate for the centered difference formula (CD) since the Jacobian is a block tridiagonal matrix with two coupling blocks on the corners. The removal of the two coupling blocks gives a block tridiagonal matrix which is costly to invert. This choice of a preconditioner motivates the fourth choice, that is, taking the preconditioner to be the block diagonal of the Jacobian matrix. Just like in the previous case, this does not work for the CD differentiation.

The second preconditioner is derived from the block diagonal matrix ( $\tilde{F} \frac{\partial H}{\partial x} \tilde{F}^*$ ) where  $\tilde{F} = (F \otimes I_m)$ . As mentioned before, the pre-image of this preconditioner is the block diagonal of a block-diagonally heavy matrix.

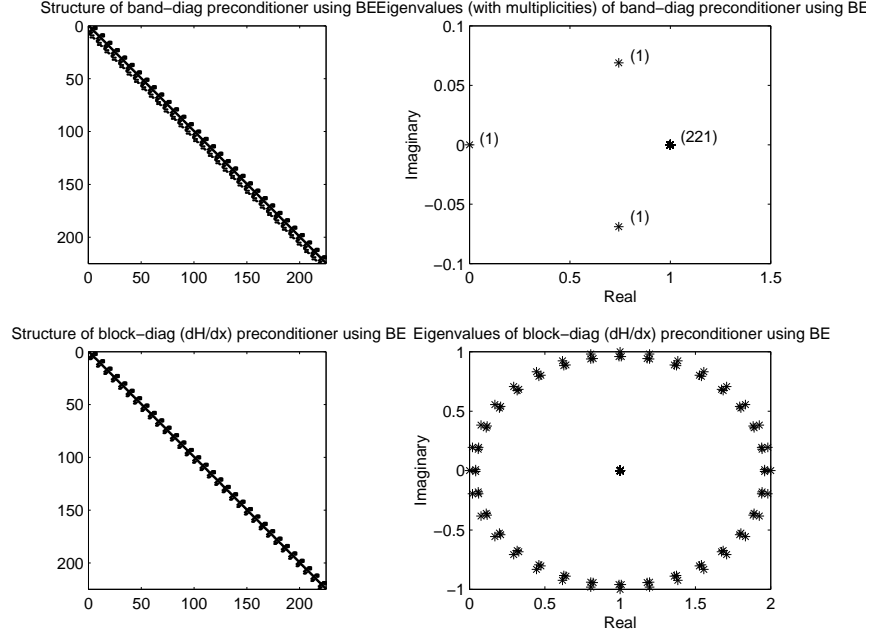
The third preconditioner considered here is  $\text{diag}(g_0, g_1, \dots, g_{n-1})$  where  $g_0, g_1, \dots, g_{n-1}$  are the Jacobians of  $f_0, f_1, \dots, f_{n-1}$ .

Finally, the identity preconditioner (i.e. no preconditioner) is investigated here.

To test these preconditioners, Jacobians were generated from the Circuit Problem in Section 2.4.3. The Generalized Minimum Residual Method was used to solve the linear systems with a randomly-generated right hand side.

### 2.4.5 Spectral Analysis

The pictures of the structure of two of the preconditioners discussed above shows that both preconditioners are “simple”, e.g., solving linear systems with these matrices is easy. The previous table shows that the band-diag ( $\frac{\partial H}{\partial x}$ ) preconditioner gives the smallest number of iterations for all types of differentiators that were tried with it. The picture of the eigenvalues of band-diag preconditioner using BE shows a cluster of 221 eigenvalues at  $(1, 0)$  and the rest very close to  $(1, 0)$ . This shows that band-diag is a very good preconditioner. However, this is not the case of the block-diag ( $\frac{\partial H}{\partial x}$ ) preconditioner. It gives a big number of iterations and the picture of the eigenvalues using BE, shows clusters away from  $(1, 0)$ . Therefore, the band-diag technique produces an effective preconditioner.



### 3 Harmonic-Balance Methods

Harmonic-balance methods differ from time-domain methods in a fundamental way. Harmonic balance approximates the solution using a linear combination of sinusoids. Since steady-state responses are periodic, harmonic balances approximate the solution very naturally.

#### 3.1 Introduction

Suppose we approximate the solution  $v(t)$  of (1) with a truncated Fourier series expansion

$$v(t) \approx u(t) = \sum_{k=-K}^K A_k e^{ik\omega t},$$

where  $A_k = A_{-k}^*$ . Thus there is a correspondence between  $u(t)$  and the vector

$$u = \begin{bmatrix} A_{-K} \\ A_{-(K-1)} \\ \vdots \\ A_{-1} \\ A_0 \\ A_1 \\ \vdots \\ A_{K-1} \\ A_K \end{bmatrix}.$$

Note that the derivative of  $q(t)$  can be represented similarly:

$$\frac{d}{dt}q(t) = \frac{d}{dt} \sum_{k=-K}^K q_k e^{ik\omega t} = \sum_{k=-K}^K ik\omega q_k e^{ik\omega t}.$$

Thus we can represent the derivative as

$$\begin{bmatrix} -i\omega K q_{-K} \\ -i\omega(K-1)q_{-K-1} \\ \vdots \\ -i\omega q_{-1} \\ 0 \\ i\omega q_1 \\ \vdots \\ i\omega(K-1)q_{K-1} \\ i\omega K q_K \end{bmatrix} = \Omega \begin{bmatrix} q_{-K} \\ q_{-K-1} \\ \vdots \\ q_{-1} \\ q_0 \\ q_1 \\ \vdots \\ q_{K-1} \\ q_K \end{bmatrix}, \quad \text{where } \Omega = \text{diag} \begin{bmatrix} -i\omega K \\ -i\omega(K-1) \\ \vdots \\ -i\omega \\ 0 \\ i\omega \\ \vdots \\ i\omega(K-1) \\ i\omega K \end{bmatrix}.$$

We define

$$H_{FD}(X) = \sqrt{n}(I_m \otimes F_n^*)(f() \otimes I_n)(I_m \otimes F_n)X + \sqrt{n}(I_M \Omega)(I_m \otimes F_n^*)(q() \otimes I_n)(I_m \otimes F_n)X.$$

We can interpret this as transforming the Fourier coefficients  $X$  into a time-domain waveform vector via the Fourier matrix  $(I_m \otimes F_n)$  and a stride permutation. Then  $n$  copies of  $f()$  and  $q()$  are evaluated at these waveform vectors. The resulting time-domain residuals are then converted back into the frequency domain.  $m$  copies of the differential operator  $\Omega$  are applied to the spectra from  $q()$ .

### 3.2 Circuit Example

The following is an example of a circuit with 6 waveforms:

$$\begin{aligned} u(t) - A_A \cos(\omega_A t) &= 0 \\ I_R - \alpha_F + I + cc \frac{d}{dt} \cdot (v_{cc} - v) + \frac{(v_{cc} - v)}{rc} &= 0 \\ I_F - \alpha_R \cdot I_R - \frac{(s - v_{ee})}{re} - ce \cdot \frac{d}{dt} s &= 0 \\ lc \frac{d}{dt} I - (v_{cc} - v) &= 0 \\ I_F - I_s \cdot (e^{rVt(u-s)} - 1) &= 0 \\ I_R - I_s \cdot (e^{rVt(u-v)} - 1) &= 0 \end{aligned}$$

with  $lc = 9.2 \cdot 10^{-9}$ ,  $cc = 39 \cdot 10^{-12}$ ,  $rc = 1000$ ,  $\alpha \cdot F = 0.98$ ,  $\alpha \cdot = 0.25$ ,  $ce = 10^{-6}$ ,  $re = 8000$ ,  $v_{cc} = 15, v_{ee} = -15$ ,  $I_s = 10^{-16}$ , and  $rVt = 40$ . We rewrite this system of nonlinear ODE to

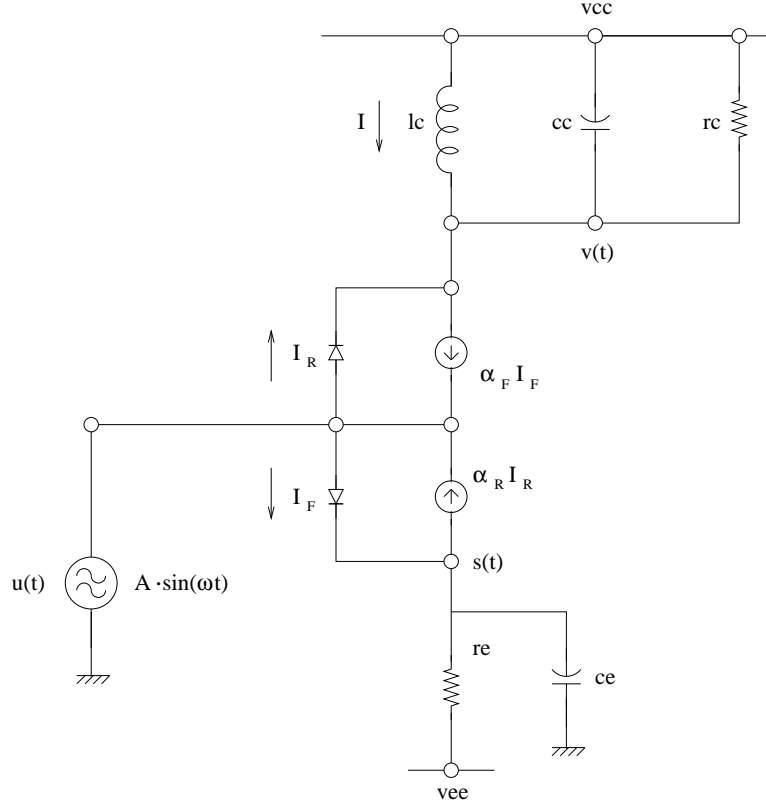
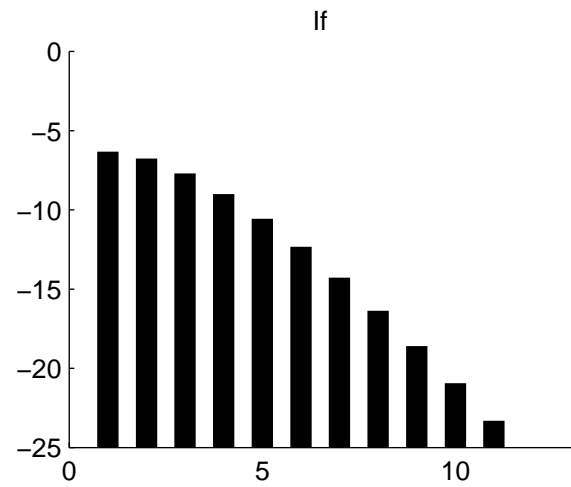
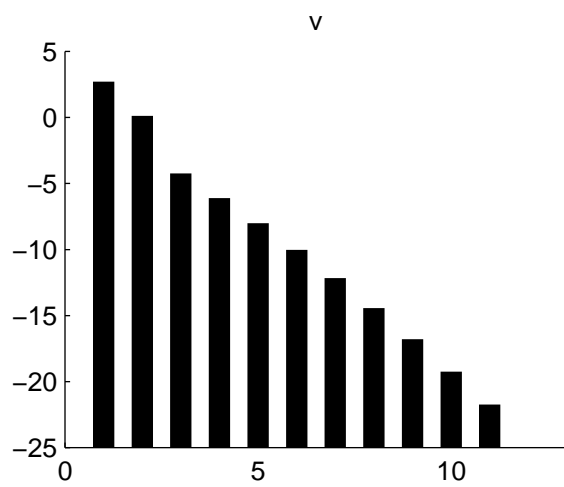


Figure 4: Circuit Schematic of Amplifier

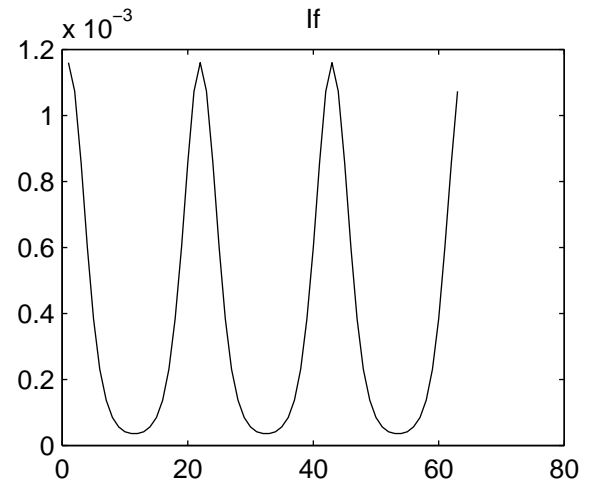
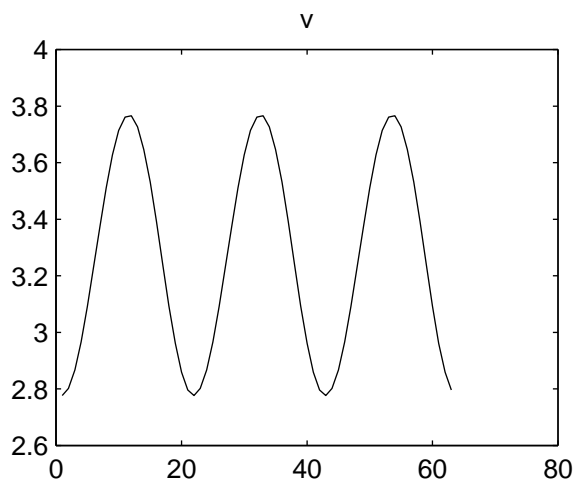
fit in (1):

$$f : \begin{bmatrix} u \\ v \\ s \\ I \\ I_F \\ I_R \end{bmatrix} \rightarrow \begin{bmatrix} u \\ I_R - \alpha \cdot I_F + I + \frac{(vcc-v)}{lc} \\ I_F - \alpha \cdot I_R - \frac{(s-vee)}{re} \\ -(vcc-v) \\ I_F - Is \cdot (e^{rVt \cdot (u-s)} - 1) \\ I_R - Is \cdot (e^{rVt \cdot (u-v)} - 1) \end{bmatrix}, \quad q : \begin{bmatrix} u \\ v \\ s \\ I \\ I_F \\ I_R \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ cc \cdot (vcc - v) \\ -ce \cdot s \\ lc \cdot I \\ 0 \\ 0 \end{bmatrix}, \quad b(t) = \begin{bmatrix} \cos(\omega_A t) 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

We need to define the Jacobian of  $H$  to solve this equation using Newton's method. The most computationally expensive part is the factorization of  $J_H$ . The Jacobian also will be obtained by the form in the previous section. We'll introduce preconditioners in the next section. Although Newton's method with linesearch is a strong zero-finding algorithm, it has been our experience that without having a good initial value, the algorithm fails to converge. We can see that finding a good initial guess is a key issue in determining its convergence. The responses for the waveforms  $v$  and  $I_F$  are shown both in the time and frequency domains, in the following figure. The behavior of the real solution for  $I_F$  shows the obvious nonlinearity. The response in the frequency domains is obtained by the following way: Since the solution vector  $X$  is of the odd centered form, we only take lower half vectors (including the DC term). Then the logarithm of the absolute values of these vectors indicates the following graph when  $k=10$  ( i.e.  $n = 21$ ).



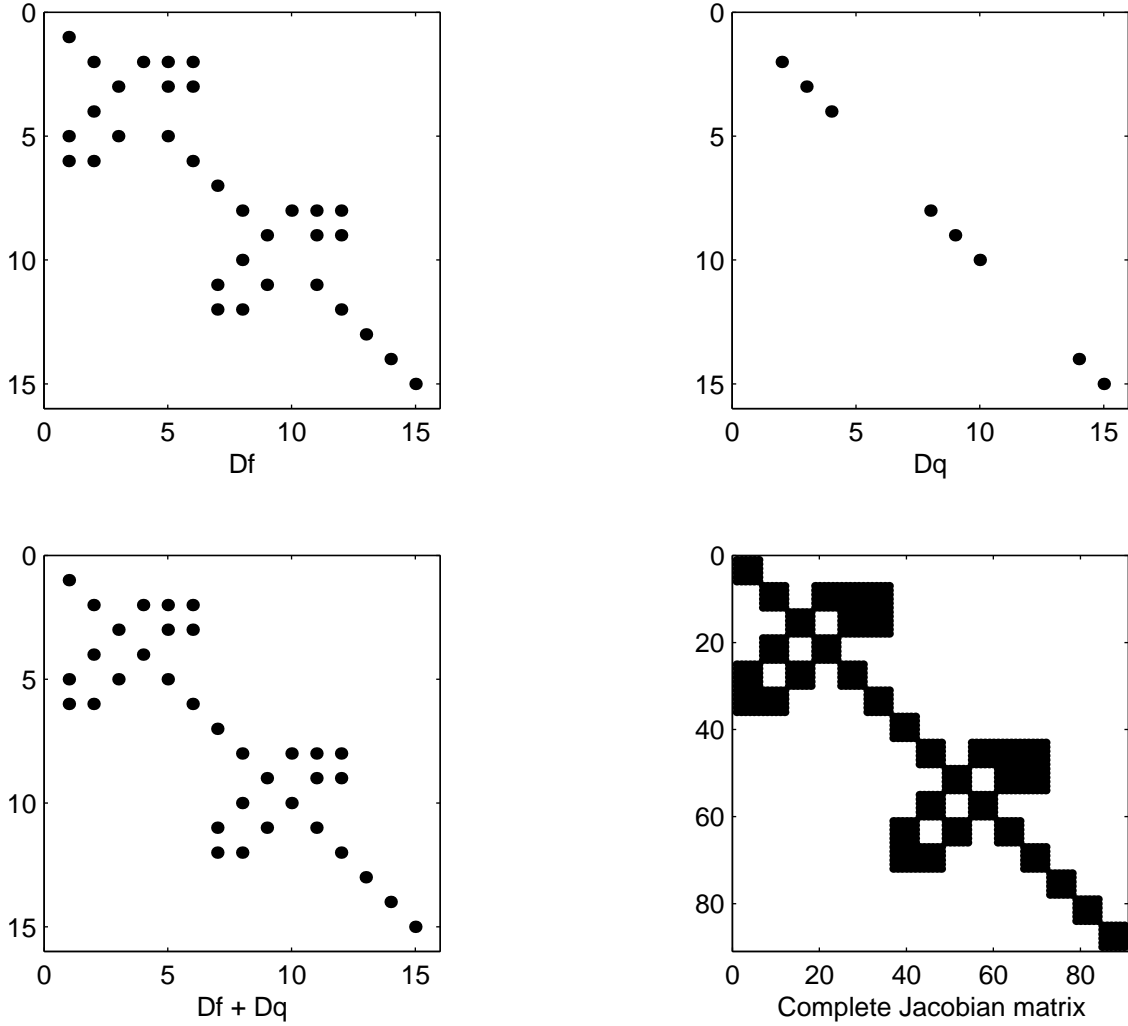
responses in frequency domain



responses in time domain

### 3.3 Preconditioning

We now consider two types of preconditioning in the frequency domain. In the cricuits example,  $m = 6$  and  $n = 21$ .  $Df$ ,  $Dq$ ,  $Df + Dq$  and the complete Jacobian matrix have the following sparsity patterns:

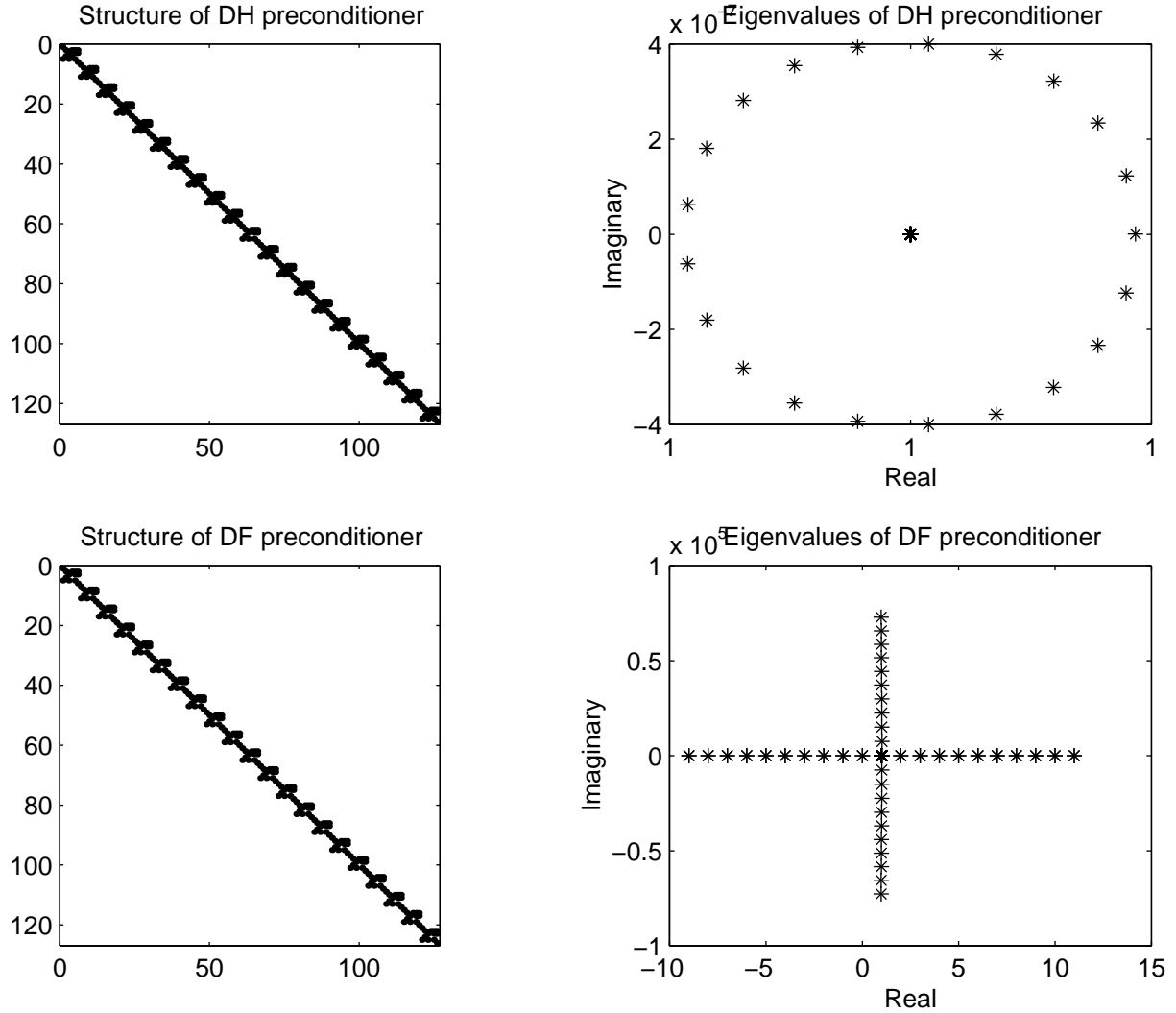


An obvious choice for a pre-conditioner would be to simply drop the  $Dq$  terms. That is, define

$$\tilde{J} = (I_m \otimes G^*)(Df() \otimes I_n)(I_m \otimes G).$$

An apply of this pre-conditioner requires a direct solve with  $(Df() \otimes I_n)X$ , which is related to  $(I_n \otimes Df())$  by a stride permutation ( $Df$  version). This is fast and easy to implement. Unfortunately, computational experience shows that this pre-conditioner works poorly.

Another construction is much stronger. In this case we pick the matrix of  $n$  diagonal  $m \times m$  blocks of  $DH$  as the pre-conditioner ( $DH$  version). From the following figure, it is easy to see that the  $DH$  pre-conditioner is much better than  $Df$  version.



To get a solution with relative residual  $1e-11$ , *Df* version converged at about the  $63^{rd}$  iteration, however, *DH* version converged at about the  $3^{rd}$  iteration.

## 4 Future Work

The authors would like to investigate further effective preconditioners for both the finite-difference method and harmonic balance. In particular, they wish to pursue preconditioning the Jacobian matrix arising from the centered-difference formula. Also, they will consider non-uniform time steps in the discretization of the ODE. Adaptive refinements will be explored and upper bounds on the error estimates will be given.

## References

- [1] R. Burden, J. Faires *Numerical Analysis*, Brooks/Cole Publishing, 1997.
- [2] R. Melville *A Proposal for Effective Preconditioning of Multi-Tone Harmonic Balance Systems*

- [3] R. Tolimieri, M. An, C. Lu *Mathematics of Multi-dimensional Fourier Transforms*, Springer-Verlag, 1993.
- [4] R. Melville, P. Feldmann, J. Roychowdhury, "Efficient Multi-tone Distortion Analysis of Analog Integrated Circuits," Proc. of CICC, 1995.
- [5] "Harmonic Balance Analysis of Semiconductor Equations", dissertation, B. Troyanovsky, R. Dutton, Stanford EE, 1995.
- [6] P. Feldmann, R. Melville, D. Long, "Efficient Frequency Domain Analysis of Large Nonlinear Analog Circuits," Proc. of CICC, 1996.
- [7] H. Brachtendorf, G. Welsch, R. Laur, "A Simulation Tool for the Analysis and Verification of the Steady State of Circuit Designs," Int. J. of Circuit Theory and App., Vol. 23, pp. 311–323, 1995.
- [8] M. Roesch, K. Antreich, "Schnelle stationaere Simulation Nichtlinearer Schaltungen im Freuenzbereich", AEUE, Vol. 46, No. 3, pp. 168–176, 1992.
- [9] K. Kundert, J. White, A. Sangiovanni-Vincentelli, "Steady-state Methods for Simulating Analog and Microwave Circuits," Kluwer, Boston, MA, 1990.
- [10] R. Gilmore, M. Steer, "Nonlinear circuit analysis using the method of harmonic balance—a review of the art," Int. J. on Microwave and Millimeter Wave Comp. Aided Eng., Vol. 1, Jan. 1991.
- [11] S. Maas, "Nonlinear Microwave Circuits," Artech House, 1988.
- [12] V. Rizzoli, A. Costanzo, P. Ghigi, F. Mastri, D. Masotti, C. Cecchetti, "Recent Advances in Harmonic-Balance Techniques for Nonlinear Microwave Circuit Simulation," AEUE, Vol. 46, No. 4, pp. 286–296, 1992.
- [13] V. Rizzoli, C. Cecchetti, A. Lipparini, F. Fatri, "General-purpose harmonic balance analysis of nonlinear microwave circuits under multitone excitation," IEEE Trans. on Microwave Theory and Tech., vol. MTT-36, pp. 1650–1660, Dec. 1988.
- [14] Y. Saad and M. Schultz, "GMRES: A Generalized Minimal Residual Method for Solving Nonsymmetric Linear Systems," SIAM J. Sci. Stat. Comput., Vol. 7, pp. 856–869 (1986).
- [15] A. Ushida, L. Chua, "Frequency domain analysis of nonlinear circuits driven by multi-tone signals", IEEE Trans. on Circuits and Sys., vol. CAS-31, pp. 766–779, Sep. 1984.
- [16] W. Hackbush, "Iterative Solution of Large Sparse Systems of Equations," Springer Verlag App. Math. Sci., Vol. 95, Springer Verlag, 1994.
- [17] R. Freund, G. Golub, N. Nachtigal, "Iterative solution of linear systems," Acta Numerica, pp. 57–100, 1991.